Reactive displays: improving man-machine graphical communication

by JOHN D. JOYCE and MARILYN J. CIANCIOLO
Research Laboratories
General Motors Corporation
Warren, Michigan

INTRODUCTION

The on-line graphic representation and solution of problems is opening the door to new and exciting computer applications. Continuous man-machine interaction via graphic consoles makes feasible the solution of entirely new classes of problems. This expanding use of computer graphics is requiring improved techniques of man-machine communication and graphic data management. At the General Motors Research Laboratories, we have had the opportunity since 1962 for considerable experimentation in a man-machine environment. From these experiments new ideas have evolved about how to improve the two-way information flow between the console user and the computer model of his problem. A fundamental concept is the reactive display, which supplies immediate graphical response to the actions of a man at a console. We have found that reactive displays provide a good basis for interaction between the man and the individual phases of his problem.

A problem to be solved is that of providing a good foundation in graphic displays. The characteristics of this foundation must be such that console users can easily do productive work at a console for several hours per day. These console users in general will have no computer training and will have no inclination or desire to become encumbered by the intricacies of conventional computer work. These users may be designers, draftsmen, electrical engineers, or project planners who merely want to communicate with a console in order to get a job done. Our experiences have demonstrated that using alphanumeric language statements to produce graphic displays is not the best environment for the types of console users mentioned above. An alphanumeric language is an abstraction of a problem which is harder to understand than a pictorial abstraction of a problem. For example, a project planner could describe all the activities of a project, their durations and the precedence relationships which exist among the activities, solely with an alphanumeric language. However, this same information is much better described with a PERT diagram.

Let us consider some of the steps involved in using a computer to help solve a problem. For a segment of a problem, a sequence of one or more procedures is requested, several data items are entered as inputs to the procedures and results are produced. Certain sets of procedures have specific relationships. Only certain sets of data items satisfy requirements for particular procedures. All other data items can be considered to be background or contextual information when a particular procedure is in control. A foundation for graphic displays must provide the facilities for representing and allowing the selection of individual procedures, actions, and data items pictorially. It is also essential that the graphic system dynamically represent changing relationships among various sets of procedures and data items. Many errors can be eliminated by only allowing selection of syntactically correct inputs and by providing dynamic feedback to ease the correction of other human errors.

Let us assume that a project planner wants activity A to have a longer duration, a different description, and different precedence relationships; also, that a PERT diagram is displayed on the console and that a "Change Activity" procedure is in control. All precedence lines are made non-selectable by a light-pen. However, all precedence lines remain displayed so that the activities are viewed in context with the precedence lines. The user places the light-pen on the screen. Activity B becomes brighter. Although this selection is syntactically correct, the user has made a human error and adjusts the light-pen position until Activity A becomes brighter. Satisfied with the immediate feedback the user removes the pen from the

screen and now a different set of items can be changed, i.e., (a) the description of A, (b) the duration of A, and (c) the lines of precedence of A. And so the process continues with different sets of display entities changing to reflect the ever-changing status of the problem.

Objectives

Three aspects of a graphic system will be presented. (See Figure 1.) First, we shall discuss some of the human factors of computer graphics, i.e., what is the best problem-solving environment for the man at a console. Second, we shall describe in detail the graphic programming system. They key elements of this system, its internal structure, and the advantages and disadvantages of certain implementation techniques will be presented. By graphic programming systems we mean the software support which is the linkage between the display hardware and the applications programmer. Third, we shall consider the facilities as seen by the applications programmer. It is his job to build upon the graphic programming system a graphic language which will best communicate with the console user.

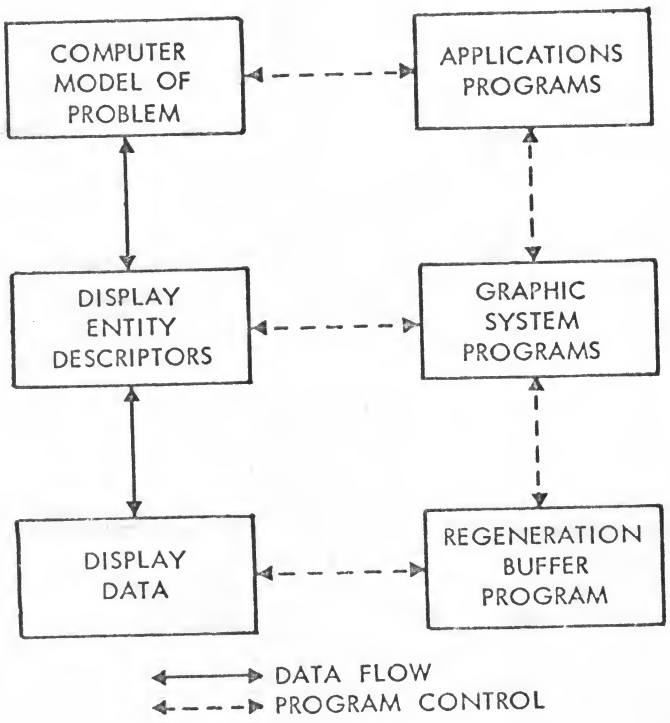

Figure 1 – Graphic system programs as an interface

In this paper we shall limit ourselves to a discussion of the tools we have designed and implemented to interface between the applications programmer and the graphic console user. We shall make no attempt to discuss the design or implementation of a graphical language. Many kinds of man-machine communications media could be built atop the graphical system we discuss in the following pages.

Definition of terms

A few terms will be defined here for clarity. These definitions are not suggested as general definitions but only for their use in the remainder of the paper.

A *user* or *console user* will refer to a person who manipulates console controls such as a light-pen function buttons, typewriter, etc., to do productive functions such as mechanical or electrical design or parts layout. This person usually has no knowledge of computer hardware or software and is only interested in the console as a tool to get his own job done.

An *applications programmer* shall mean the person who provides the software to do specific productive jobs. This software is built upon the graphic system software and the time-sharing system software.

A *graphical language* is considered to be one in which communication at the CRT is carried out by continuous interaction with pictorial representations of a problem. In such a language very little use is made of alphanumerics, except for labels and a few key control words. A language which generates pictures from executing groups of alphanumeric statements is not considered to be a graphical language.

The *human factors* of graphics refers to all aspects of communication with a man at a console. The object here is to provide the maximum comfort and convenience of operation for a console user and to increase efficiency by keeping good response time and eliminating obvious sources of errors of syntax. These points will be discussed in more detail under the section on Man-Machine Communication Techniques.

Man-machine communication techniques

No graphical communication system will be an effective problem-solving tool unless it is natural and convenient to use. Consequently, at the GM Research Laboratories we have given a great deal of attention to optimizing user satisfaction. The fundamental goal has been to bring the man ever closer to direct communication with a computer. This communication should be in a conversational graphical language which the man understands.

Discrete problem elements and procedures

The beginnings were primitive: a display was thought of as one large picture, a kind of visual feedback. Language statements were fed into a card reader by the user and executed. The results were displayed on the console screen. Gradually, console use expanded to communication more directly in graphics.

It was not evident in the beginning that working toward a solution of a graphical problem involved the manipulation of individual data elements. For example, a mechanical parts designer will work with an individual line when he is changing the design. An electronic circuit designer will change a circuit by working with an individual component and its relationship to other components.

Focusing the user's attention

Another aspect of external communication was not apparent during initial attempts at a graphic system. This is the need to gain the attention of the console user by directing him to a certain portion or portions of the display. There are many ways of achieving this: possible methods are intensification, blinking, modulation of size, or changing the color of portions of the display.

After investigating several possibilities, we concentrated on the use of intensification for highlighting parts of a display. For example, the four boundary lines of a surface may be brightened while the interior lines are dimmed. This technique of varying intensification has proven to be an important piece of feedback to the console user. We call the preceding example "static" intensification because an item remains intensified throughout the current display. Static intensification can be used meaningfully in other ways, besides highlighting. For example, user attention can be focused on suggested choices of control words by brightening them, while dimming the rest of the display. Shading to give a three-dimensional effect is also useful in optimizing the user's understanding of a display. This technique, however, requires enough intensity levels so that the transition between them is continuous and imperceptible.

Dynamic intensification is a more important form of brightening. An item, a line and its label for example, is displayed at a very high level of brightness during the time that the pen is pointing at it. (See Figure 2). Whenever the pen moves to a void area, nothing is intensified. If the pen is pointed at another part of the display, that part will be intensified. Intensification of the selected item is feedback that tells the console operator exactly what he is pointing at. When the operator is satisfied with his selection, he removes his pen from the screen.

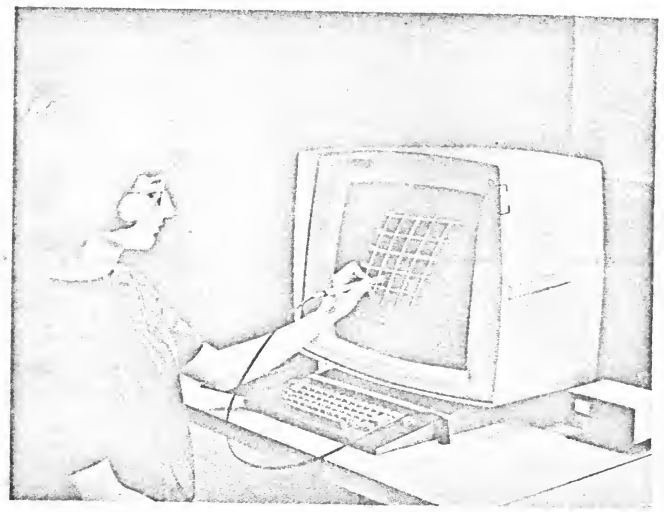

Figure 2 – Dynamic intensification of a display entity

Restricting selectability of display items

Another important mechanism in the restricting of selectability to only certain items or classes of items in the display. It is obvious that some background information such as borders or grid lines may be displayed, but is never selectable. Perhaps not so obvious is that, frequently in reactive displays, classes of items should be made non-selectable. The console user might, for example, wish to execute a line smoothing procedure. He will probably be required to indicate the line in which he is interested. At this point in time, then, only lines should be candidates for selection. All other kinds of items should be "disabled" for detection. Further, only lines should be giving a feedback response in the form of dynamic intensification.

This "selective disabling" aids the operator in recognizing his displays as a composition of various sets of items. In addition, he will be less confused when trying to make a selection. Only a small subset of items can be selected, even though other information is still displayed. Thus, the selectable items are kept is context with the rest of the display. The combination of selective disabling and dynamic intensification provides the means of conveying the syntax of a graphical language to the console user.

Other techniques

There are other human factors involved in console use, which are not a direct concern of the graphical systems designer. They merit attention because these external criteria must be met with a minimum of effort by the graphical language implementer.

In production design work, for example, console users are sometimes at work for several hours or more. This prolonged use clearly indicates the need to minimize the number of manipulations required to solve a problem, e.g., the number of "pokes" at the screen. It is the job of a good graphical system to interface with the graphical language in meeting this important requirement.

The graphical language can further enhance user understanding by providing the ability to

(a) change the scale or size of the data,
(b) change the direction of view,
(c) rotate or otherwise transform data,
(d) label items on the screen.

When the user sees these actions carried out in continuous motion, under his control, his visualization of the problem is improved. A good graphical system lays the groundwork for implementing these capabilities. It does this by providing the means of effortless communication between the applications programmer and the graphic device.

All the graphical techniques just described help to improve the all important information flow, the conversation, between the computer and the man. Each small part of the display now becomes a variable under the control of the man. He can follow the progress of his problem and analyze intermediate results. He can, in fact, change the flow of activity to an entirely new direction. He is continuously reacting to the everchanging display, and equally important, the display is reacting to him.

A characteristic of the above graphical communication techniques is the high burst rate of information flow. A console user can receive information with his eyes at rates as high as 4.3×10^6 bits/second.[1] He can transmit decisions back to the computer at rates as high as 10^2 bits/second. These high rates, although they cannot be sustained, do require an internal data structure which provides efficient display handling and good response time.

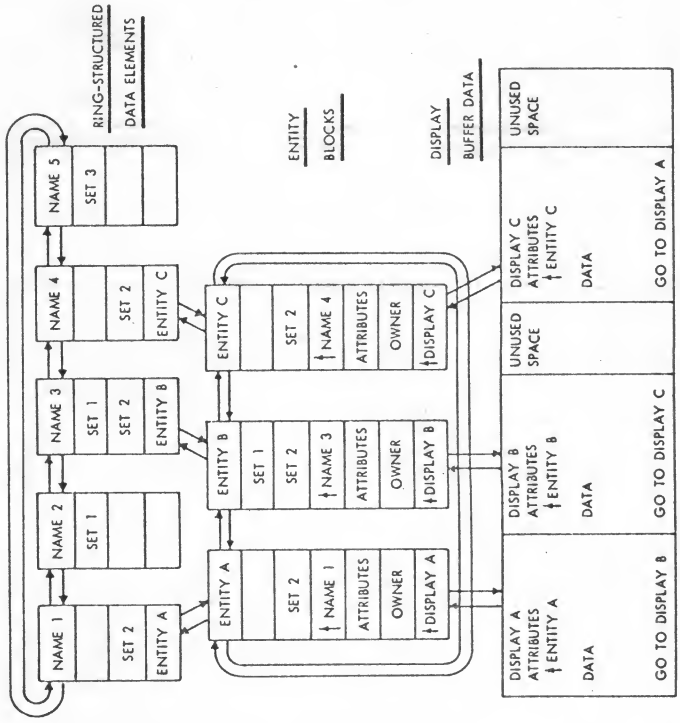

Figure 3 – Elements of reactive displays

Graphic programming system

Elements of a reactive display

The preceding paragraphs have described some of the external requirements met by the graphic programming system at the GM Research Laboratories. It is now appropriate to consider the basic internal structures that can help meet these needs most efficiently. (See Figure 3.)

As mentioned earlier, a display can be thought of as one large picture. This means display changes require working with the entire display. Another approach is to change a display by manipulating only a small part of it. This involves a decomposition of the large display into smaller displays called "entities." An entity may be defined as a distinct collection of displayable data to which the user may wish to make a unique reference. It represents some meaningful part of the model of the problem. The display data comprising an entity may be any combination of vectors, characters, or points. It may be scattered anywhere on the screen.

Each entity has display attributes assigned to it, which determine its external characteristics. It is described by a level of beam intensity and the attribute of selectability or non-selectability. Further, to

speed internal manipulation, each entity may be classified by its membership in a set(s). The membership of each entity in zero or more sets determines its relationship to the other entities in the display. In addition, the relations between display entities reflect the relations of the data items to which they correspond. To further aid in rapid data management, a two-way direct link has also been established between every displayed data item or control word and its display buffer counterpart.

The use of these basic structural ideas has helped to create a graphic system which can quickly and effectively handle the flow and manipulation of graphical data. The system affords a natural means of communication with the console user and, at the same time, ease and flexibility to the applications programmer.

Structure for entity descriptors

The need for flexible graphic control has impressed upon us the importance of certain internal requirements. No matter what type of display system is being implemented, it is useful to have blocks of entity descriptors, each of which describes an entity and its properties. Blocks in current use are linked together in a ring structure. (See Figure 3.) In the same way, unused blocks are linked together in a "free space" ring. They can be obtained quickly when another entity is to be added. Each active block contains a pointer to the previous entity and to the next entity, information about the entity attributes, and the size and display buffer location of the entity data. (See Figure 3.) It is assumed that the display data itself resides in the display buffer memory. The order of the entity descriptor ring matches the order of regeneration in the display buffer. This implicit information is used when adding or deleting entities.

It might be argued that a separate ring structure to describe display entities is unnecessary. Display characteristics could be included in a ring structure which described the data elements themselves. However, our experiments have indicated that a great deal of CPU processing time, as well as time to access peripheral storage devices,* was required to search the data ring structures. A less time-consuming method must be used to manipulate display entities since online response time is a major consideration in graphical work.

*To minimize the amount of CPU memory used most of the large data structure was maintained on high-speed drums. Searching a ring structure which threaded through the data often involved bringing many "pages" of data into CPU memory. Access time to a given data item was a function of the depth at which it was buried in the ring structure.

Data elements which correspond to display entities are generally a small subset of those which describe the entire problem. Thus a small amount of memory can contain a ring structure that describes the display characteristics of displayed data elements and the displayed control words, which have no data element counterpart.

The compactness of entity descriptor information offers advantages in both a paging and non-paging environment.[2,3] With paging and ring structures the descriptor blocks can be contained within one or two data pages. If it is assumed that pages will remain in memory based on frequency of use, then the blocks will nearly always be accessible without retrieval from a peripheral device. If the blocks are "paged out" while the console user is thinking, their compactness will permit rapid retrieval. The blocks will remain in memory so long as frequency of graphic attentions is high.

In the non-paging environment the small size of the entity descriptor ring makes it possible to keep it in the CPU. Many additions and deletions of blocks can therefore be carried out quickly and thus, maintain good response time for the console operator.

Control of execution sequences

It is important to give to the applications programmer the ability to enable or to inhibit the use of hardware features. These might include a light-pen, an alphanumeric keyboard, or programmed function keys. A single interrupt handling code in the graphics system dispatches control to the appropriate place when these features are activated. Transfer of control is carried out by having the user supply an "owner" (in the form of a program name or statement label) for each hardware feature and each display entity. The owner is recorded in an appropriate control block. When an enabled feature or entity is selected, control is transferred to the corresponding "owner." This eliminates the need for applications programs to decode attention information.

Linking display entities and problem elements

Problems of search and table look-up are eliminated by a direct two-way link between a display entity and the data element to which it corresponds. The applications program assigns to each entity a name or pointer which either has a distinct meaning in the computer model of the problem, or is a direct pointer to its associated data element. A graphics system program provides another pointer which is guaranteed to be unique. This pointer is directly linked to an entity for subsequent changes to or deletion of that

entity. When an entity is selected, both pointers are returned to the owner program. Experiments have shown that a direct two-way link between a display entity and a data element maximizes efficiency in manipulating the display.

Display hardware capabilities

In order to make the two-way pointer system work most effectively, the display controller hardware should have certain logical capabilities. Without CPU intervention the controller should be able to "name" every new entity as the pen passes over that entity. It can do this by physically moving the name or label of the detected entity to a fixed buffer location. When the final selection is indicated by the man removing his pen from the screen, the graphics system program needs only to read the contents of that fixed buffer location. The pointer to the corresponding data element or control word is thus available without any further processing. To make the concepts of reactive displays most effective in a time-sharing environment, it is desirable to reduce the number of CPU interrupts to a minimum.

We found this could be best accomplished by making additional demands of the graphic hardware. Specifically, we needed the ability to perform dynamic intensification of display entities without CPU intervention. We also needed to handle display attributes from within the controller. This required hardware control of display intensity and the enable-disable status of entities.

These capabilities imply that the display controller can:

(a) detect light and remember this information,
(b) do a conditional transfer on the remembered detection,
(c) move the name of the detected entity to a specified location,
(d) control beam intensity with a single buffer order,
(e) enable or disable graphical data with a single buffer order.

(See Figure 4.)

Associative relationships among entities

A graphical display is simply a visual representation which helps the console user to understand a computer model of his problem. Within this model relationships exist between one data item and other data items. In a job scheduling model, for example, a given job is related to all the jobs which must be completed before that given job is begun. In the model of a design problem two points are related by a distance. Another kind of relationship exists in which one data element is related to others because they are members of the same set. For example, all lines in a drawing belong to one set, while all lines which belong to Surface A comprise a sub-set of the original set.

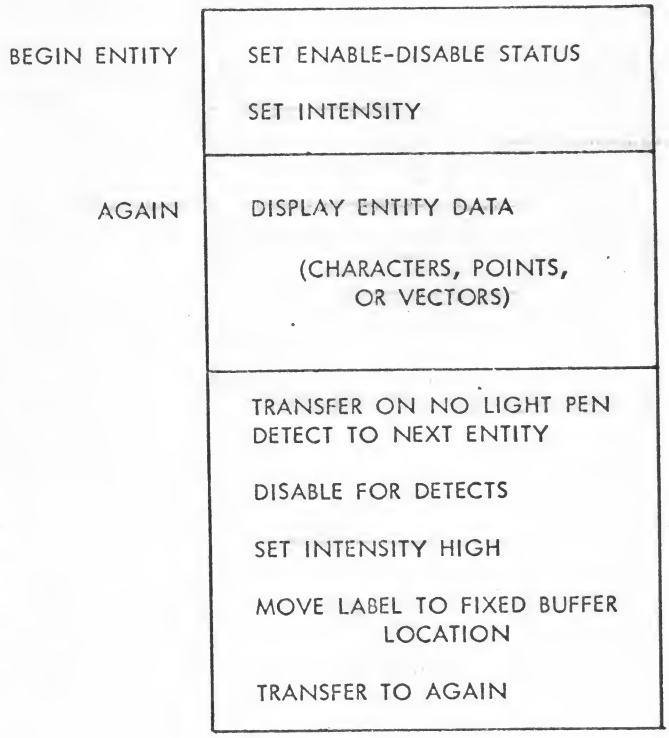

Figure 4 – Entity structure in display buffer

The need to manipulate data items based on associative relationships is illustrated by the fact that whole new languages have been written expressly for that purpose.[4,5] We have found that in manipulating graphical display entities, it is at least necessary to work with sets of entities or logical combinations of sets of entities. Suppose, for example, that at some point in time only a "line entity" which lies on "Surface A" is an acceptable screen selection. The entities in this set need to be rapidly enabled for operator selection, while all others should be disabled. At some other point in time all "point entities" on a "Centerline" should be brightened, or only the control words "LINE SMOOTHING" or "SURFACE EVALUATION" may be applicable. The display system must be able to react rapidly to such requested changes in display characteristics.

From the above examples we see that the relationships among display entities are simply a reflection of the relationships which exist among their corresponding data items. We have found it sufficient to record

the set membership of entities in the entity descriptor blocks. Tests on an experimental system indicated that the assignment of each entity to membership in one or more sets was convenient and adequate to handle display manipulations based on associative properties.

Handling the associative relationships among displayed items on an entity basis means that the graphical system does not need to work with the entire data structure. Since the data set for the entity descriptors is almost always smaller than the total data set for the model of a problem, there is a real advantage to the above method. In addition, display characteristics are tied directly to the associative relationships. Display changes are thus speeded up because the same program which searches for the requested set of entities can change the appropriate display characteristics at the same time. In a data environment where the associative relationships among data items are not considered at all, handling sets of display entities at the graphical system level provides a capability not otherwise possible.

The applications programs supply the names of the sets (if any) of which each entity is to be a member. The graphic system saves these names and the information about the set membership of each entity. As additional entities become members of existing sets, they are marked as such. A new set is created whenever an entity has membership in one or more sets which are currently unknown to the graphic system. Sets may become empty when entities are deleted, or whenever set membership of one or more entities has changed. Empty sets are automatically purged from the system when the additional space is required for new sets.

An incident matrix was selected to record the membership of entities in various sets. The expected maximum number of entities (150-200), the maximum number of sets (25-30), and the distribution of entities among sets indicated that an incident matrix would conserve storage and computer time. One row of the matrix is stored with each entity descriptor.

The disadvantage of this method is that it is difficult to accommodate more than an arbitrary number of sets at a given time. A ring structure could be used to describe associative relationships, but would require more computer time and storage.

Logical combinations of sets of entities may be deleted or have their properties changed by one request from the applications program. Any combination of sets which can be expressed by "AND," "OR," and "NOT" operations can be handled. Some of the more frequently used combinations are of the following types:

(a) entities which are members of Set A "OR" Set B,
(b) entities which are members of Set A "AND" Set B,
(c) entities which are members of Set A "OR" Set B and "NOT" of Set C.

These unlimited combinations of sets have provided facilities for making display changes conveniently and efficiently. The ability to make such changes provides a syntactical basis for graphical displays.

Display buffer management

No system restrictions have been placed on either the size or number of display entities which can exist at one time. The physical size of the regeneration buffer provides an upper limit. When an entity is deleted from the display buffer, the space it occupied is given back to the graphic system. The regeneration cycle is altered only to the extent that pertinent transfer addresses are changed. The entity previous to the deleted entity now transfers to the entity which followed the deleted entity. This method of deletion has several advantages:

First, there is no need for the system to maintain a copy of the regeneration data in CPU memory. Second, no processing time is required to recreate the regeneration cycle and store it as a contiguous block in buffer memory. Third, the amount of data transferred to the display buffer is minimal.

To summarize, entity deletion as described above saves CPU memory space, CPU processing time, and I/O time. One disadvantage of this method is that variable length blocks of unused space remain in the buffer. Since these holes may or may not be large enough to accommodate a new entity, efficiency of space usage becomes an item of concern.

A graphic system using variable length entities and this form of deletion was written and sampled to test fragmentation of the buffer memory. The results indicated that only a moderate percentage of space was wasted on holes interspersed among entities. Extensive testing showed that the number of holes did not usually exceed 10% of the maximum number of entities. Even with complex applications, the number of entities does not normally exceed 150-200 within the framework of current CRT sizes and regeneration buffers. This means that a small CPU memory table can accommodate all the information needed to manage buffer space allocation.

Facilities for applications programmers

The capabilities of the graphic system described in this paper are available to the applications pro-

grammer for a high-level language (PL/I). The following can be easily accomplished:

(a) Creation of new display entities
(b) Deletion of entities
(c) Modification of entity properties
(d) Modification of entity data
(e) Modification of properties of logical combinations of sets of entities
(f) Deletion of logical combinations of sets of entities
(g) Enabling of hardware functions (typewriter keyboard, function buttons, light-pen for positional input)
(h) Temporary inhibition of all interrupts and re-enabling for interrupts again

Each one of the functions listed above is available by an individual subroutine call in PL/I. The programmer provides only X,Y data and attribute information when he creates an entity. He need not be concerned about the management of space in the display buffer. All display control orders and logic are generated automatically by a graphic systems routine. I/O generation and transmission are also done automatically.

When the graphic system has information to present to the problem program, CPU control is transferred to the appropriate "owner" location. This location was previously indicated to the system for each display entity and hardware function. The system also makes available information about the interrupt type and, if applicable, the X,Y position of the pen.

The applications programmer is thus freed from the task of interfacing his computer model of a problem with the display hardware. The two-way information flow is carried out quickly and efficiently at a systems level. The result is good response time and economical CPU operation.

SUMMARY

The principles of reactive displays can be used in many different computer console arrangements. At General Motors Research Laboratories three graphic systems have been written, all of which utilize the concepts described in this paper. (See Appendix.) In spite of internal dissimilarities and hardware differences, these three systems provided nearly identical capabilities to the applications programmer.

The reactions of both the applications programmers and console users have been enthusiastic. The applications programmer appreciates the flexibility and efficiency of the system. He can easily communicate with the graphic hardware and has access to specific information about user respones. Console users indicate that they especially like the dynamic intensification of entities and fast response time. They also appreciate the selective enabling of only meaningful entities. From the standpoint of internal efficiency, our current thinking is that graphic systems based on concepts of reactive displays offer maximum speed and ease of data manipulation.

Appendix

Three graphical systems using the principles discussed in this paper have been implemented at the General Motors Research Laboratories. The main features in these systems are listed in Table I.

	SYSTEM A	SYSTEM B	SYSTEM C
CPU	IBM 7094	IBM 360/50	IBM 360/67
LANGUAGE	NOMAD	PL/I	PL/I
SOFTWARE ENVIRONMENT	MULTIPROGRAMMING	TIME-SHARING	TIME-SHARING
DISPLAY CONSOLE	DAC-I CONSOLE	IBM 2250-I	IBM 2250-III
REGENERATION BUFFER	WITHIN CPU	ONE PER CONSOLE	SHARED BUFFER (IBM 2840-II)
DISPLAY CONTROLLER LOGIC	ALMOST NONE	LIMITED AMOUNT	MODERATE AMOUNT
POINTING DEVICE	VOLTAGE PENCIL	LIGHT PEN	LIGHT PEN

Table I—System configurations

The first system was built upon the software and hardware of the DAC-I System.[6,7,8] The IBM 2250-I consoles used in the second system were used for program checkout until the IBM 2250-III consoles and 2840-II controller were installed. This latter combination of hardware included, in addition to the Graphic Design Feature, orders for a conditional transfer on light-pen tip switch open and orders to control display intensity.

ACKNOWLEDGMENT

The authors wish to express their appreciation to Edwin L. Jacks and Fred N. Krull for their help in the development of the ideas presented in this paper.

REFERENCES

1 H. JACOBSON
The informational capacity of the human eye
Science, vol. 113, b, pp. 292-293 1951

2 J G DENNIS
Segmentation and the design of multiprogrammed computer systems

J. of Association for Computing Machinery vol 12 no 4 pp 589-602 October 1965

3 B W ARDEN B A GALLER T C O'BRIEN F H WESTERVELT
Program and addressing structure in a time sharing environment
J. of Association for Computing Machinery vol 13 no 1 pp 1-16 January 1966

4 G G DODD
APL-a language for associative data handling in PL/I
1966 Proc FJCC vol 28 pp 677-684

5 L G ROBERTS
Graphical communication and control languages
Second Congress on the Information System Science Spartan Books Washington D C 1964

6 E L JACKS
A laboratory for the study of graphical man-machine communication
1964 Proc FJCC vol 26 pp 343-350

7 B HARGREAVES J D JOYCE G L COLE et al
Image processing hardware for a man-machine graphical communication system
1964 Proc FJCC vol 26 pp 363-386

8 M P COLE P H DORN C R LEWIS
Operational software in a disc-oriented system
1964 Proc FJCC vol 26 pp 351-362

BIBLIOGRAPHY

1 S H CHASEN
The introduction of man-computer graphics into the aerospace industry
1965 Proc FJCC vol 27 pp 883-891

2 T E JOHNSON
Sketchpad III: A computer program for drawing in three dimensions
1963 Proc SJCC vol 23 pp 347-353

3 W H NINKE
Graphic 1-A remote graphical display console system
1965 Proc FJCC vol 27 pp 839-846

4 M D PRINCE
Man-computer graphics for computer-aided design
Proc. of IEEE vol 54 no 12 pp 1698-1708
December 1966

5 D E RIPPY D E HUMPHRIES
MAGIC-A machine for automatic graphics interface to a computer
1964 FJCC vol 27 pp 819-830

6 R STOTZ
Man-machine console facilities for computer aided design
1963 Proc SJCC vol 23 pp 323-328

7 I E SUTHERLAND
Sketchpad: A man-machine graphic communication system
1963 Proc SJCC vol 23 pp 329-346